TLM.open: a SystemC/TLM Front-end for the CADP Verification Toolbox

Claude Helmstetter
INRIA Grenoble Rhône-Alpes / Vasy
655, avenue de l'Europe, F-38330 Montbonnot St Martin, France

1 Introduction

The development of embedded systems starts more and more by the design of abstract models written in SystemC/TLM [18, 6]. These models allow the simulation of the embedded software before the hardware RTL descriptions are available, and are used as golden models for hardware verification. The verification of the SystemC/TLM models is an important issue, since a error in the model can mislead the system designers, or reveal an error in the specification.

The OSCI provides an open-source simulator for SystemC/TLM and a library SCV to ease test generation. However, the OSCI does not provide tools for formal verification. Moreover, while the SystemC specification allows many schedulings for a given test case, the OSCI simulator exhibits always the same scheduling. Thus, even if an execution leads to the expected result, another execution with a different scheduling may be erroneous. To find this kind of bugs, many publications have experimented the use of model checking

In order to apply model checking to a SystemC/TLM program, the usual approach relies on the translation of the SystemC/TLM code to a formal language for which a model checker is available. Defining and implementing this kind of translation is a difficult task because SystemC/TLM programs can contain arbitrary C++ code.

We propose another approach that suppress the translation effort. Basically, an explicit model checker (e.g., CADP) must be able to execute *transitions* and to store *states*. Given a SystemC/TLM program, we execute the transitions using `g++` and the OSCI library, and we ask the user to provide additional functions to store the current program state. These additional functions represent generally less than 20% of the size of the origianl model, and allows to apply all CADP tools to the SystemC/TLM program itself.

The remainder of this abstract is organized as follows. Section 2 gives an overview of the previous works. Section 3 presents the existing CADP toolbox and next describes the new `TLM.open` front-end. The features and performances of `TLM.open` are evaluated in section 4. Section 5 concludes this abstract.

2 Previous works

In order to provide formal verification for SystemC/TLM programs, two approaches have been investigated: stateless model checking of a SystemC/TLM program, and translation of a SystemC/TLM program to a language for which a stateful model checker is available.

The stateless model checking techniques that have been implemented for SystemC/TLM programs [8, 13, 1], give interesting results for small and medium sized industrial examples. However, they can be applied only to bounded test scenarios.

For programs that do not terminate, a second approach has been investigated. The idea is to translate the SystemC/TLM program to be verified to another language, and then verify the translated program using an existing stateful model checker. This approach has first been applied to the RTL level SystemC descriptions [2, 7].

Many translations and languages have been proposed for the validation of transactional models, like [17], which translates SystemC/TLM programs into finite state machines (FSM), or like [12], which describes abstraction techniques and a translation from SystemC/TLM to labeled Kripke structures. Like our approach, most of these translations are manually, a notable exception being the LusSy tool chain [16], which automatically translates TLM models into synchronous automata with variables; it provides some simple abstraction techniques (e.g., abstract address representation). The LusSy tool chain has been connected to many model checkers, including symbolic model checkers based on BDD or SAT. Some small examples have been successfully verified, but industrial examples face the state space explosion problem.

The state space explosion problem appears chiefly because TLM models are mainly asynchronous. Indeed, after each transition, there are many valid scheduling choices that should be explored. It is therefore suitable to use the model checkers for asynchronous programs, as these model checkers have been specifically optimized to fight state space explosion arising from asynchrony. For example, the SPIN model checker uses partial orders to reduce state spaces; a translation of TLM to Promela is described in [20], allowing the use of SPIN to verify TLM models.

Also, we recently proposed a translation of TLM to LOTOS [9, 19] that enables verification of the benchmark of [20] for a slightly greater number of processes than using SPIN. [4] presents an application of our TLM to LOTOS translation for an industrial case study; this paper shows that some properties can be verified but this approach requires too much manual work.

```
R ← {initial state} //set of remaining states
E ← ∅ //set of explored states
while (∃x ∈ R) do
  foreach transition x → y do
    if (y ∉ E ∪ R) then R ← R ∪ {y}
  R ← R \ {x}
  E ← E ∪ {x}
end while
```

Figure 1: Basic algorithm for explicit model checking.

3 Model checking of SystemC/TLM programs

3.1 The CADP toolbox

CADP ("Construction and Analysis of Distributed Processes") [5] is a toolbox for the validation of communication protocols and distributed systems. CADP provides numerous tools, among them:

- step-by-step, interactive, and random simulators
- a model checker that generates an explicit representation of the state space (i.e., the *Labeled Transition System (LTS)* of the system)
- property checker for various temporal logics
- equivalence checker and LTS minimization tools.

The usual entry point for CADP is the language LOTOS. The ISO standard LOTOS [10] (*Language Of Temporal Ordering Specification*) is a process algebra used to describe asynchronous concurrent processes communicating and synchronizing by rendezvous on gates. This language is well-suited for designing communication protocols.

However, to model embedded systems at the transaction level, engineers of industrial companies prefer to use SystemC/TLM. One reason is that SystemC/TLM provides natively all the useful features, like shared memory and transactional communication channel. Another reason is that a SystemC/TLM program is mainly C++ code, so engineers can learn SystemC/TLM quickly, and existing C code can be easily reused.

Hopefully, the CADP toolbox is modular and each tool can be connected to various front-ends, using the OPEN/CÆSAR interface [3]. A front-end for the CADP toolbox consists in a C library providing the operators required by the model checker itself.

Figure 1 describes an algorithm for explicit model checking, without any optimizations. An enhanced version of this algorithm is implemented in CADP. The following operators are required to run this algorithm, and must be implemented in the front-end:

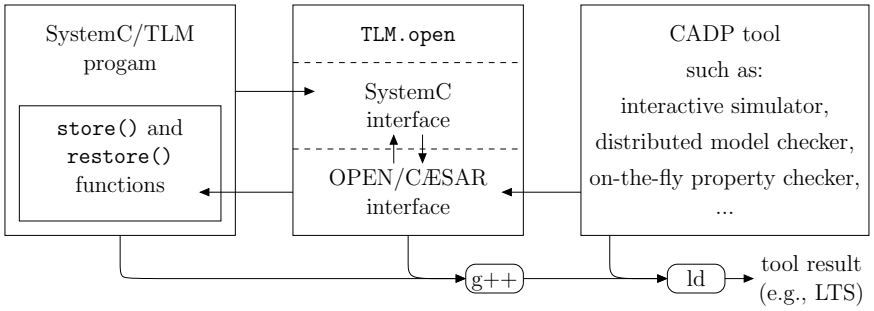

Figure 2: Overview of the verification framework.

- generation of the initial state
- enumeration and execution of the transitions starting from a given state
- efficient storage of a state (requires comparison and hash functions).

3.2 A SystemC/TLM front-end for CADP

Figure 2 provides an overview of our verification framework based on a new SystemC/TLM front-end for CADP, called `TLM.open`. The `TLM.open` front-end consists in a C/C++ library implementing two interfaces. Firstly, `TLM.open` provides and implements a subset of the OSCI SystemC library, including modules, events, TLM ports, and processes (`SC_METHOD` only). Secondly, `TLM.open` implements the OPEN/CÆSAR interface.

In order to use `TLM.open`, the user must implements a few callback functions for each SystemC module. These functions are needed by `TLM.open` in order to implement the OPEN/CÆSAR interface.

- `size_t size() const`: number of bytes needed to store a copy of the SystemC module
- `size_t alignment() const`: specify whether padding bytes are needed
- `void store(char *dest) const`: store the current state of the module in `dest`
- `void restore(const char *src)`: restore the state of the module according to the copy stored in `src`.

The `store()` function must generate a canonical representation, so state comparison can be done using `memcmp()` and hash functions can be generated automatically.

Implementing the functions `size()` and `alignment()` requires generally one line of code for each. The `store()` function implementation contains two lines of code per module member on average; same for the `restore()` function. Implementing these functions requires some manual work, but less than translating

the whole model to another language. Moreover, given a SystemC parser like Pinapa [15], it should be possible to generate these functions automatically. Such a tool would be simple compared to an automatic SystemC to automata translator like LusSy [16].

When is used with the model checker of CADP, the result is an LTS with two kinds of transitions:

- `TE` transitions indicate that some time has elapsed; the parameters give the duration and the list of triggered events.
- `EXEC` transitions represent the execution of a SystemC process; the parameters provide the name of the elected process, the inputs of the process if this process has called the special `rand()` function of `TLM.open`, and the outputs generated using the `puts()` function.

4 Examples

4.1 The `chain` benchmark

We evaluate our new front-end on the benchmark proposed in [20] and reused in [9]. This benchmark consists of a chain of interrupt transmitter modules, whose length is parametrized by n. Modules communicate trough transactions, and processes synchronize with events. Figure 3 presents the SystemC original benchmark for $n = 1$. To increase n, one adds a transmitter module between the last transmitter and the sink module. There are always $n + 2$ processes (functions named `compute` and `process`) and $n + 1$ events (private attribute e of each module).

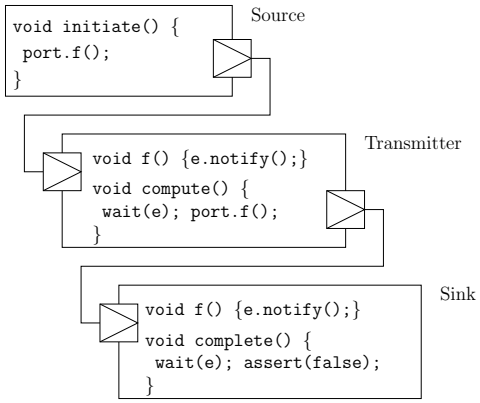

Figure 3: The `chain` benchmark for $n = 1$

Table 1 presents the results for the generation of the full LTS, using a Linux machine with 4 GB of memory. For comparison, [20] succeeded to verify this benchmark up to $n = 15$ (47 seconds), and [9] succeeded up to $n = 19$ (8293

Table 1: Results of the experiments using `TLM.open`

$n =$	3	7	11	15	19	20
LTS generation	5 s	5 s	5 s	14.6 s	403 s	1151 s
state number	62	1022	16,382	262,142	4,194,302	8,388,606
state number after minimization	47	767	12,287	196,607	3,145,727	n.a.

seconds for $n = 19$; 60.2 seconds for $n = 15$). The results of table 1 show a significant speedup compared to the previous approaches based on the translation to Promela or LOTOS. The efficiency of `TLM.open` can be explained by two points:

- One transition in the LTS corresponds exactly to one SystemC transition (i.e., the execution of a process between two `wait` statements.) There are no additional transitions used to mimic the behavior of the SystemC scheduler.
- The memory size of a state is kept as small as possible, allowing the model checker to store more states.

4.2 Verification of a simple timer

This subsection illustrates the features provided by `TLM.open` by showing how it can be applied to a simple but realistic example. We consider a timer with two registers FREQ and ACK.

- Writing a non-null value to FREQ starts the timer
- When enabled, the timer generates an interruption periodically
- Writing to ACK acknowledges the interruption
- Writing 0 to FREQ stops the timer.

We have four SystemC/TLM models of this timer at our disposal. The first one (`v1`) comes from the SimSoC project [11]; the second (`v2`) is the first with a bug-fix; the third and the forth have been provided by an engineer and a PhD student (`v3` and `v4`).

In order to verify the first SystemC/TLM model, which contains 80 lines of code, we had to write 17 additional lines of code to implement the `store()` and `restore()` functions. The timer verification requires to design an environment modeling the commands generated by the embedded software. For this example, we decided to describe the environment in LOTOS because `TLM.open` allows the user to merge SystemC/TLM code with LOTOS code, and LOTOS is well-suited for nondeterministic programs.

Firstly, we applied on-the-fly property checking. The property checker of CADP [14] revealed an error in the first version: for some particular scheduling,

the timer could generate an interruption after it has been stopped. A counterexample was automatically exhibited, allowing us to fix the bug. Another minor bug was found in the third version `v3`.

Secondly, we tried the equivalence checker of CADP. We generated the LTS of each SystemC/TLM model, we hided the internal transitions, and we minimized the LTS according to branching equivalence. CADP provided us with the guarantee that the versions `v2` and `v4` are bisimilar modulo branching equivalence. It means that if one contains an error, the other contains the same error. As expected, the first and third versions are not equivalent, because they contains distinct errors.

5 Conclusion

We have presented a new framework for the verification of SystemC/TLM programs. Our new SystemC/TLM front-end avoids the need to translate the whole SystemC/TLM program to another language. Compared to approaches based on manual translation, the verification using `TLM.open` is much more simple: there are less lines of code to write and the engineer do not need to learn a new modeling language. Moreover, `TLM.open` allows to scale up a little further than the previous works. Thanks to the numerous tools of CADP, it is now possible to check complex properties, and to test the equivalence of two SystemC/TLM programs.

We believe that the `store()` and `restore()` functions can be generated using a SystemC parser. The result would be a tool-chain as automatic as LusSy, but much more simple to develop and maintain.

We have started experiments on a large industrial case study. As explained in [4], the most difficult task to verify a SystemC/TLM program is to extract an abstract model that is simple enough to be formally verified. Our current goal is to integrate `TLM.open` is in the design flow in such a way that this task becomes simple and safe.

References

[1] Nicolas Blanc and Daniel Kroening. Race analysis for SystemC using model checking. In *Proceedings of ICCAD 2008*, pages 356–363. IEEE, 2008.

[2] Rolf Drechsler and Daniel Große. Reachability analysis for formal verification of systemc. In *DSD*, pages 337–340. IEEE Computer Society, 2002.

[3] Hubert Garavel. Open/cæsar: An open software architecture for verification, simulation, and testing. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *LNCS*, pages 68–84, Berlin, March 1998. Springer. Full version available as INRIA Research Report RR-3352.

[4] Hubert Garavel, Claude Helmstetter, Olivier Ponsini, and Wendelin Serwe. Verification of an Industrial SystemC/TLM Model using LOTOS and CADP. In *7th ACM-IEEE International Conference on Formal Methods and Models for Codesign MEMOCODE'2009*, Cambridge, MA United States, 2009.

[5] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Cadp 2006: A toolbox for the construction and analysis of distributed processes. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification CAV'2007 (Berlin, Germany)*, volume 4590 of *LNCS*, pages 158–163. sv, July 2007.

[6] Frank Ghenassia, editor. *Transaction-Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*. Springer, June 2005. ISBN 0-387-26232-6.

[7] D. Große and R. Drechsler. CheckSyC: an efficient property checker for RTL SystemC designs. In *ISCAS*, volume 4, pages 4167–4170, May 2005.

[8] Claude Helmstetter, Florence Maraninchi, Laurent Maillet-Contoz, and Matthieu Moy. Automatic generation of schedulings for improving the test coverage of systems-on-a-chip. *FMCAD*, pages 171–178, 2006.

[9] Claude Helmstetter and Olivier Ponsini. A comparison of two SystemC/TLM semantics for formal verification. In *Proceedings of the 6th ACM-IEEE International Conference on Formal Methods and Models for Codesign MEMOCODE'2008*, June 2008.

[10] ISO/IEC. Lotos — a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1989.

[11] Vania Joloboff and Claude Helmstetter. SimSoC: A SystemC TLM integrated ISS for full system simulation. In *Asia Pacific Conference on Computer Architecture and Systems*, Macao Macao, 2008.

[12] Daniel Kroening and Natasha Sharygina. Formal verification of SystemC by automatic hardware/software partitioning. In *MEMOCODE '05*, pages 101–110. IEEE, 2005.

[13] Sudipta Kundu, Malay Ganai, and Rajesh Gupta. Partial order reduction for scalable testing of SystemC TLM designs. In *DAC '08: Proceedings of the 45th annual conference on Design automation*, pages 936–941, New York, NY, USA, 2008. ACM.

[14] Radu Mateescu and Damien Thivolle. A model checking language for concurrent value-passing systems. In Jorge Cuellar and Tom Maibaum, editors, *Proceedings of the 15th International Symposium on Formal Methods*

FM'08 (Turku, Finland), number 5014 in lncs, pages 148–164. sv, May 2008.

[15] Matthieu Moy, Florence Maraninchi, and Laurent Maillet-Contoz. Pinapa: an extraction tool for systemc descriptions of systems-on-a-chip. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 317–324, New York, NY, USA, 2005. ACM.

[16] Matthieu Moy, Florence Maraninchi, and Laurent Maillet-Contoz. LusSy: an open tool for the analysis of systems-on-a-chip at the transaction level. *Design Automation for Embedded Systems*, 2006. special issue on SystemC-based systems.

[17] B. Niemann and Ch. Haubelt. Formalizing TLM with communicating state machines. In *FDL'06: Forum on Specification & Design Languages*, pages 285–292, September 2006.

[18] Open SystemC Initiative. *SystemC v2.2.0 Language Reference Manual (IEEE Std 1666-2005)*, 2006. `http://www.systemc.org/`.

[19] Olivier Ponsini and Wendelin Serwe. A schedulerless semantics of TLM models written in SystemC via translation into LOTOS. In Jorge Cuellar and Tom Maibaum, editors, *Proceedings of the 15th International Symposium on Formal Methods FM'08 (Turku, Finland)*, number 5014 in LNCS. sv, May 2008.

[20] Claus Traulsen, Jérôme Cornet, Matthieu Moy, and Florence Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In *14th Workshop on Model Checking Software SPIN*, July 2007.